数 据 结 构

——C 语言描述

主　编　王志雄　田小梅

副主编　曾志峰　刘本军　杨海英

连智峰　葛　蓉　方全文

北京工业大学出版社

图书在版编目（CIP）数据

数据结构：C 语言描述 / 王志雄，田小梅主编 .—北京：北京工业大学出版社，2006.8
ISBN 7 – 5639 – 1694 – 6

Ⅰ. 数… Ⅱ. ①王…②田… Ⅲ. ①数据结构—高等学校：技术学校—教材②C 语言—程序设计—高等学校：技术学校—教材 Ⅳ. ①TP311.12②TP312

中国版本图书馆 CIP 数据核字（2006）第 092367 号

数据结构——C 语言描述

主编 王志雄 田小梅

※

北京工业大学出版社出版发行

邮编：100022 电话：(010) 67392308

各地新华书店经销

徐水宏远印刷厂印刷

※

2006 年 8 月第 1 版 2006 年 8 月第 1 次印刷

787 mm × 1 092 mm 16 开本 15.75 印张 387 千字

印数：0001 ~ 5000 册

ISBN 7 – 5639 – 1694 – 6/G·843

定价：28.00 元

前　言

近年来，在全球信息化浪潮的推动下，我国的信息技术产业在较短的时间内得到了迅猛发展。一方面，计算机的应用已深入到社会的各个领域，计算机已成为当今科学技术现代化和管理现代化不可缺少的重要工具；另一方面，计算机从业人员较少、专业素质较低，社会对计算机专业人才的需求日益迫切。

“数据结构”是计算机学科的核心课程之一，是重要的专业基础课，是提高学生软件设计水平和程序编写能力以及学习后续课程所必需的基础。在计算机学科的各个领域中，几乎都要用到数据结构。不仅如此，数据结构中所包含的内容及其思想精髓，使得它已成为很多非计算机专业开设的选修课或辅修课。

在本书的编写过程中，充分考虑我国高等职业教育的培养目标和教学现状，力求体现高职高专的教学特点，以“理论够用，注重实用”为原则，对传统的数据结构课程中理论性较强、实用性较少、复杂度较高的教学内容进行了简化和忽略；对常用的、基本的和主要的数据结构与算法进行了详细的介绍。本书主要内容包括：线性表、栈、队列、串、数组、树、二叉树、图、文件等数据结构及常用的排序和查找算法。

本课程教学的目的是：通过对一些常用数据结构的学习，揭示数据之间内在的逻辑关系，讨论数据在计算机中的存储表示，分析施加在数据上的各种运算（操作）方法并评价其优劣。从而，学生能熟练地掌握各种常用数据结构的基本特性、算法实现及其性能好坏，并能在实际应用中根据求解问题的性质和要求，选择合理的数据结构，设计高效的运算方法，控制算法的时空复杂度。通过该课程的学习，一方面，为学生学习后续计算机专业课程提供必要的知识准备；另一方面，进一步提高学生的软件设计水平和程序编写能力，使学生编写的程序结构清楚、准确易读、符合软件工程的规范。如果说高级语言程序设计课程对学生进行了结构化程序设计（程序抽象）的训练，那么数据结构课程将要培养学生的数据抽象能力，使学生具备面向数据编程的基本思维。

本书采用C语言作为数据结构和算法的描述语言。在编写中力求概念通俗易懂、简洁明了；内容由浅入深、循序渐进；实例引用贴切、图文并茂；操作易于理解、注重实用。为便于教学，每章后面还配备了习题。

本书既可作为高等职业技术院校、高等专科学校、成人高等学校及本科院校所属的二级职业技术学院计算机类专业或信息类相关专业的教材，也可作为非计算机专业学生选修课或辅修课的教材，还可作为计算机应用人员和工程技术人员的参考书。

本书由王志雄、田小梅担任主编，由曾志峰、刘本军、杨海英、连智峰、葛蓉、方全文担任副主编。参加本书编写的有（以姓氏笔画为序）：方全文、王志雄、田小梅、江敏、刘本军、刘红霞、杨海英、连智峰、郦丽、侯廷刚、秦天增、桑莉君、葛蓉、曾志峰。全书由王志雄统稿和定稿。

由于编写时间仓促，加之编者水平有限，书中难免有不当之处，恳请读者批评指正。

王志雄

2006年6月

目　录

第1章 绪 论

“数据结构”是随着电子计算机的产生和发展而发展起来的一门较新的计算机学科。计算机系统分为软件系统和硬件系统两大部分，软件是用各种程序设计语言编写的。为了能够以最少的成本、最快的速度、最好的质量开发出合乎要求的软件，最重要的是建立起合理的软件体系结构和程序结构，设计有效的数据结构。因此，作为软件设计人员就必须了解如何组织各种数据在计算机中的存储、传递和转换。这正是数据结构这门课所要研究的问题。

数据结构作为一门独立的课程最早是在美国的一些大学开设的，1968 年美国唐·欧·克努特教授开创了数据结构的最初体系，他所著的“计算机程序设计技巧”第一卷“基本算法”是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作。从 20 世纪 60 年代末到 20 世纪 70 年代初，出现了许多大型程序，软件也相对独立，结构程序设计成为程序设计方法的主要内容，人们越来越重视数据结构，认为程序设计的实质是对确定的问题选择一种好的结构，加上设计一种好的算法。从 20 世纪 70 年代中期到 20 世纪 80 年代初，各种版本的数据结构著作相继出现。我国大多数院校从 1978 年开始，先后开设的“数据结构”，目前已成为计算机各专业的核心课程之一，许多非计算机专业也把“数据结构”作为必修和选修课程。

数据结构在计算机科学中是一门综合性的专业基础课。数据结构的研究不仅涉及到计算机硬件（特别是编码理论、存储装置和存取方法等）的研究范围，而且和计算机软件的研究有着更密切的关系，无论是编译程序还是操作系统，都涉及到数据元素在存储器中的分配问题。在研究信息检索时也必须考虑如何组织数据，使查找和存取数据元素更为方便。因此，可以认为数据结构是介于数学、计算机硬件和计算机软件三者之间的一门交叉课程，在计算机科学中，数据结构不仅是一般程序设计（特别是非数值计算的程序设计）的基础，而且是设计和实现编译程序、操作系统、数据系统及其他系统程序和大型应用程序的重要基础。

数据结构所讨论的有关问题，最先是为解决系统程序中的具体技术而出现在操作系统和编译技术之中的，20 世纪 60 年代中期数据结构的前身称为表处理语言（List Processing Language）。典型的表处理语言有：

20 世纪 50 年代初设计的 SLIP 系统（简单表处理语言）；

1954—1959 年设计的 IPL－V 系统（信息处理语言）；

1959—1960 年设计的 LISP 系统（表处理语言）；

1962 年初设计的 SNOBOL 系统（串处理语言）。

这些语言都是以数据为中心，为处理非数值问题设计的。它们适用于情报检索、信息管理等系统，例如，航空订票、事务管理等。这类问题要求用复杂的数据结构来描述系统的状态，它们的运算是实现对数据结构的访问和修改。然而，大家所熟知的数值问题则完全不同，这类问题属于在简单的数据结构基础上进行复杂的函数计算。早期美国部分大学虽然把

数据结构列为一门独立的课程，但对课程的内容并没有明确的规定。当时的数据结构几乎和图论，特别是表和树的理论是同义语。由于数据必须在计算机中进行处理，因此，人们不仅要考虑数据本身的数据性质，还必须考虑数据在计算机内的存储方式，这就扩大了数据结构的内容。随着数据库系统、情报检索系统的不断发展，在数据结构的技术中又增加了文件结构，特别是增加了大型文件的组织等方面的内容和 B 树、B^+ 树的知识。数据结构逐步成为了一门比较完整的学科。

值得注意的是，数据结构的发展并未终结，一方面，面向各专门领域中特殊问题的数据结构得到研究和发展，如多维图形数据结构等；另一方面，从抽象数据类型的观点来讨论数据结构，已经成为一种新的趋势，越来越被人们所重视。

1.1 数据结构的概念和术语

在计算机发展的初期，人们使用计算机的目的主要是处理数值计算问题。当人们使用计算机来解决一个具体问题时，一般需要经过下列几个步骤：首先要从该具体问题抽象出一个适当的数学模型，然后设计或选择一个解决此数学模型的算法，最后编出程序进行调试、测试，直至得到最终的解答。寻求数据模型的实质是分析问题，从中提取操作的对象，并找出这些操作对象之间含有的关系，然后用数据的语言加以描述。例如，求解梁架结构中应力的数学模型的线性方程组，该方程组可以使用迭代算法来求解。

由于当时所涉及的运算对象是简单的整型、实型或布尔类型数据，所以程序设计者的主要精力是集中于程序设计的技巧上，而无须重视数据结构。随着计算机应用领域的扩大和软、硬件的发展，非数值计算问题越来越显得重要。据统计，当今处理非数值计算性问题占用了 90% 以上的机器时间。这类问题涉及到的数据结构更为复杂，数据元素之间的相互关系一般无法用数学方程式加以描述。因此，解决这类问题的关键不再是数学分析和计算方法，而是要设计出合适的数据结构，才能有效地解决问题。现举例说明之。

【例 1-1】 多叉路口交通灯的管理问题。通常，在十字交叉路口只需设红、绿两色的交通灯便可保持正常的交通秩序，而在多叉路口需设几种颜色的交通灯才能既使车辆相互之间不碰撞，又能达到车辆的最大流通呢？

如图 1-1 所示的五叉路口，其中 C 和 E 为单行道，在路口有 13 条可行的通路（A→B、A→C、A→D、B→A、B→C、B→D、D→A、D→B、D→C、E→A、E→B、E→C、E→D），其中有的可以同时通行，如 A→B 和 E→C，而有的不能同时通行，如 E→B 和 A→D。那么，在路口应如何设置交通灯进行车辆的管理呢？

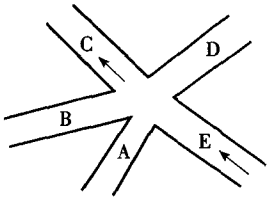


图 1-1 五叉路口示意图

通常，这类交通、道路问题的数学模型是一种称为“图”的数据结构。如图 1-2 所示，

图中每个圆圈（又称顶点）表示五叉路口上的一条通路，两个圆圈之间的连线（又称边）表示两个圆圈所代表的通路，不能同时通行。则设置交通灯的问题等价为对图的顶点的染色问题，要求对图上的每个顶点染一种颜色，并且要求有线相连的两个顶点又不具有相同颜色，而总的颜色种类尽可能少。图1-2所示为一种染色结果，数字表示交通灯的不同颜色。

（1）号色灯：A→B、A→C、A→D、B→A、D→C、E→D。

（2）号色灯：B→C、B→D、E→A。

（3）号色灯：D→A、D→B。

（4）号色灯：E→B、E→C。

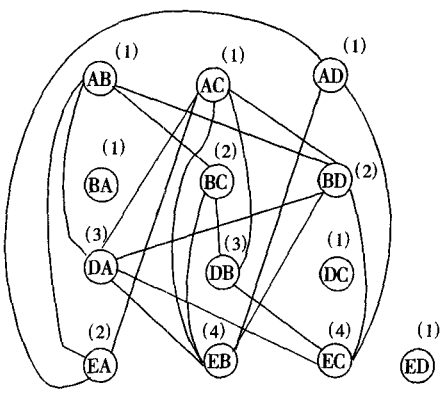


图1-2 五叉路口表示通路的图

很多问题的求解最后都转化为求解数学方程或数学方程组，即使是不需要用计算机求解的简单问题也需要一个数学模型来描述。例如，大家都熟悉的“鸡兔同笼”问题，可转化为“二元一次方程组”进行求解。又如，在房屋设计或桥梁设计中的结构应力分析计算可化解为线性代数方程组求解的问题。再如，天天看到的天气预报，它的数学模型是一个环流模式方程。

综上所述，描述这类非数值计算问题的数学模型不再是数学方程，而是诸如表、树、图之类的数据结构。通过以上讨论，可以直观地认为：数据结构是一门研究非数值计算的程序设计和计算机的操作对象之间的关系及其操作等问题的学科。

在系统地学习数据结构知识之前，先对一些基本概念和术语赋予确切的含义。

1. 数据（data）

数据是描述客观事物的数、字符以及所有能输入到计算机中被计算机程序加工处理的信息的集合。数据是计算机程序加工的“原料”，应用程序处理各种各样的数据。计算机科学中，所谓数据就是计算机加工处理的对象，可以是数值数据，也可以是非数值数据。数值数据是一些整数、实数或复数，主要用于工程计算、科学计算和商务处理等；非数值数据包括字符、文字、图形、图像、语音等。因此，对计算机科学而言，数据的含义极为广泛。

2. 数据元素（data element）

数据元素是数据的基本单位。在计算机程序中通常作为一个整体来考虑和处理。在不同的条件下，数据元素又可称为元素、结点、顶点、记录等。例如，学生信息检索系统中学生信息表中的一个记录、教学计划编排问题中的一个顶点等，都被称为一个数据元素。

有时，一个数据元素可由若干个数据项（data item）组成。例如，学籍管理系统中学生

信息表的每一个数据元素就是一个学生记录。它包括学生的学号、姓名、性别、籍贯、出生年月、成绩等数据项。这些数据项可以分为两种：一种称作初等项，如学生的性别、籍贯等，这些数据项是在数据处理时不能再分割的最小单位；另一种称作组合项，如学生的成绩，可以再划分为数学、物理、化学等更小的项。通常，在解决实际应用问题时是把每个学生记录当作一个基本单位进行访问和处理的。

3. 数据对象（data object）

数据对象是具有相同性质的数据元素的集合。在某个具体问题中，数据元素都具有相同的性质（元素值不一定相等），属于同一数据对象（数据元素类），数据元素是数据元素类的一个实例。例如，在交通咨询系统的交通网中，所有的顶点是一个数据元素类，顶点 A 和顶点 B 各自代表一个城市，是该数据元素类中的两个实例，其数据元素的值分别为 A 和 B。

4. 数据结构（data structure）

数据结构是指互相之间存在着一种或多种关系的数据元素的集合。在任何问题中，数据元素之间都不会是孤立的，在它们之间都存在着这样或那样的关系，这种数据元素之间的关系称为结构。根据数据元素间关系的不同特性，通常有下列四类基本的结构：

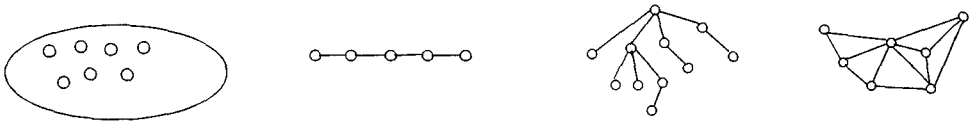
(1) 集合结构：在集合结构中，数据元素间的关系是“属于同一个集合”。集合是元素关系极为松散的一种结构。

(2) 线性结构：该结构的数据元素之间存在着一对一的关系。

(3) 树形结构：该结构的数据元素之间存在着一对多的关系。

(4) 图形结构：该结构的数据元素之间存在着多对多的关系，图形结构也称作网状结构。

上述四类基本结构的示意图如图 1-3 所示，图中每个小圆圈代表一个数据元素。



(a) 集合结构　(b) 线性结构　(c) 树形结构　(d) 图形结构

图 1-3　四类基本结构关系图

由于集合是数据元素之间关系极为松散的一种结构，因此也可用其他结构来表示。

从上面所介绍的数据结构的概念可以知道，一个数据结构有两个要素：一个是数据元素的集合；另一个是数据元素间关系的集合。在形式上，数据结构通常可以采用一个二元组来表示。

$$Data\_Structure = (D,R)$$

式中，D 是数据元素的有限集，R 是 D 上关系的有限集。

上述数据结构的形式定义仅仅是对操作对象的一种数学描述，也就是从操作对象抽象出来的一种数学模型。结构定义中的“关系”描述的是数据元素之间的逻辑关系，因此，这种形式定义又被称为数据的逻辑结构，但数据的逻辑结构不能描述计算机是如何操作数据的。研究数据结构的目的是为了在计算机中实现对它的操作，因此，人们必须研究如何在计算机中表示数据结构，即数据的物理结构。

5. 存储结构（storge structure）

数据结构在计算机中的表示（或称映像）称为数据的存储结构，又称为物理结构。

数据结构在计算机中有两种不同的表示方法：顺序表示和非顺序表示，由此得出两种不同的的存储结构，即顺序存储结构和链式存储结构。

(1) 顺序存储结构：用数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系。

(2) 链式存储结构：在每一个数据元素中增加一个存放地址的指针，用指针来表示数据元素之间的逻辑关系。

在C语言中，顺序存储结构用一维数组来描述，而链式存储结构利用指针来描述。而在实际程序设计过程中，针对一种数据的逻辑结构，到底选择哪种存储结构会影响到具体算法的设计。也就是说，在设计具体算法之前，必须先确定存储结构。在C语言中，一般使用typedef语句来为存储结构定义新的数据类型。

数据的存储结构可用以下四种基本存储方法得到。

(1) 顺序存储方法：该方法把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，结点间的逻辑关系由存储单元的位置关系来体现。由此得到的存储表示称为顺序存储结构(sequential storage structure)，通常借助程序语言的数组来描述。该方法主要应用于线性的数据结构。非线性的数据结构也可通过某种线性化的方法实现顺序存储。

(2) 链接存储方法：该方法不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系由附加的指针字段表示。由此得到的存储表示称为链式存储结构（Linked Storage Structure)，通常借助于程序语言的指针类型来描述。

(3) 索引存储方法：该方法通常在储存结点信息的同时，还建立附加的索引表。索引表由若干索引项组成。若每个结点在索引表中都有一个索引项，则该索引表称之为稠密索引(Dense Index)。若一组结点在索引表中只对应一个索引项，则该索引表称为稀疏索引（Spare Index)。索引项的一般形式是关键字和地址。关键字是能惟一标识一个结点的那些数据项。稠密索引中索引项的地址指示结点所在的存储位置；稀疏索引中索引项的地址指示一组结点的起始存储位置。

(4) 散列存储方法：该方法的基本思想是根据结点的关键字直接计算出该结点的存储地址。

上述四种基本存储方法，既可单独使用，也可组合起来对数据结构进行存储映像。同一逻辑结构采用不同的存储方法，可以得到不同的存储结构。选择何种存储结构来表示相应的逻辑结构，视具体要求而定，主要考虑运算方便及算法对时空复杂度的要求。

数据的逻辑结构、数据的存储结构及数据的运算这三方面是一个整体。孤立地去理解一个方面，而不注意它们之间的联系是不可取的。存储结构是数据结构不可缺少的一个方面，同一逻辑结构的不同存储结构可冠以不同的数据结构名称来标识。

例如，线性表是一种逻辑结构，若采用顺序存储方法，可称其为顺序表；若采用链式存储方法，则可称其为链表；若采用散列存储方法，则可称为散列表。数据的运算也是数据结构不可分割的一个方面。在给定了数据的逻辑结构和存储结构之后，按定义的运算集合及其运算的性质不同，也可能导致完全不同的数据结构。

若对线性表上的插入、删除运算限制在表的一端进行，则该线性表称之为栈；若对插入限制在表的一端进行，而删除限制在表的另一端进行，则该线性表称之为队列。更进一步，若线性表采用顺序表或链表作为存储结构，则对插入和删除运算做了上述限制之后，可分别得到顺序栈或链栈，顺序队列或链队列。

数据类型是和数据结构密切相关的一个概念。它最早出现在高级程序设计语言中，用以

刻画程序中操作对象的特性。在用高级语言编写的程序中，每个变量、常量或表达式都有一个它所属的确定的数据类型。类型显式地或隐含地规定了在程序执行期间变量或表达式所有可能的取值范围，以及在这些值上允许进行的操作。因此，数据类型（data type）是一个值的集合和定义在这个值集上的一组操作的总称。

在高级程序设计语言中，数据类型可分为两类：一类是原子类型，另一类则是结构类型。原子类型的值是不可分解的。如C语言中整型、字符型、浮点型、双精度型等基本类型，分别用保留字int、char、float、double标识。而结构类型的值是由若干成分按某种结构组成的，因此是可分解的，并且它的成分可以是非结构的，也可以是结构的。例如，数组的值由若干分量组成，每个分量可以是整数，也可以是数组等。在某种意义上，数据结构可以看成是“一组具有相同结构的值”，而数据类型则可被看成是由一种数据结构和定义在其上的一组操作所组成的。

抽象数据类型（Abstruct Data Type，ADT）是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。

抽象数据类型和数据类型实质上是一个概念。例如，各种计算机都拥有的整数类型就是一个抽象数据类型，尽管它们在不同处理器上的实现方法可以不同，但由于其定义的数学特性相同，在用户看来都是相同的。因此，“抽象”的意义在于数据类型的数学抽象特性。

但在另一方面，抽象数据类型的范畴更广，不再局限于前述各处理器中已定义并实现的数据类型，还包括用户在设计软件系统时自己定义的数据类型。为了提高软件的重用性，在近代程序设计方法学中，要求在构成软件系统的每个相对独立的模块上，定义一组数据和施于这些数据上的一组操作，并在模块的内部给出这些数据的表示及其操作的细节，而在模块的外部使用的只是抽象的数据及抽象的操作。这也就是面向对象的程序设计方法。

抽象数据类型的定义可以由一种数据结构和定义在其上的一组操作组成，而数据结构又包括数据元素及元素间的关系，因此，抽象数据类型一般可以由元素、关系及操作三种要素来定义。

抽象数据类型的特征是使用与实现相分离，实行封装和信息隐蔽。就是说，在抽象数据类型设计时，把类型的定义与其实现分离开来。

1.2 算法和算法分析

算法与数据结构的关系紧密，在算法设计时先要确定相应的数据结构，而在讨论某一种数据结构时也必然会涉及相应的算法。

算法是计算机科学和技术中一个十分重要的概念。例如，在一个大型软件系统的开发中，设计出有效的算法将起决定性的作用。通俗地讲，算法是指解决问题的一种方法或一个过程。更严格地讲，算法是由若干条指令组成的有穷序列，且满足下述几条性质。

（1）有穷性：一个算法必须在有穷步之后结束，即必须在有限时间内完成。

（2）确定性：算法的每一步必须有确切的定义，无二义性。对相同的输入算法的执行仅有惟一的一条路径。

（3）可行性：算法中的每一步都可以通过基本运算的有限次执行得以实现。

(4) 输入：一个算法具有零个或多个输入，这些输入取自特定的数据对象集合。

(5) 输出：一个算法具有一个或多个输出，这些输出同输入之间存在某种特定的关系。

算法的含义与程序十分相似，但又有区别。一方面，一个程序不一定满足有穷性。例如，操作系统，只要整个系统不遭破坏，它将永远不会停止，即使没有作业需要处理，它仍处于动态等待中。因此，操作系统不是一个算法。另一方面，程序中的指令必须是机器可执行的，而算法中的指令则无此限制。算法代表了对问题的解，而程序则是算法在计算机上的特定的实现。一个算法若用程序设计语言来描述，则它就是一个程序。

算法与数据结构是相辅相成的。解决某一特定类型问题的算法可以选定不同的数据结构，而且选择恰当与否直接影响算法的效率。反之，一种数据结构的优劣由各种算法的执行来体现。

1.2.1 算法的描述

算法需要用一种语言来描述，同时，算法可有各种描述方法以满足不同的需求，例如，一个需在计算机上运行的程序（程序也是算法）必须是严格按照语法规定用机器语言或汇编语言或高级语言编写，而一个为了人们的阅读和交流的算法可以用伪代码语言或框图等其他形式来描述，现在简单介绍一下算法描述的四种方法。

1 用自然语言描述

算法可以使用各种不同的方法来描述，最简单的方法是使用自然语言。所谓的“自然语言”指的是日常生活中使用的语言，如汉语、英语或数学语言。

【例 1-2】 想计算 1 到 N 的累加和，为简单起见，设 N 的值不大于 1000。算法可以使用自然语言描述如下：

```
S1：输入 n（要求 n<= 1000）；
S2：累加和 sum 置初值 0；
S3：自然数 i 置初值 1；
S4：若 i <= n，则重复执行：
S41：i + sum -> sum;
S42：i + 1 -> i;
S5：输出 sum，结束。
```

这就是用自然语言配合数学语言描述算法。用自然语言描述的算法通俗易懂，而且容易掌握，便于人们对算法的阅读。但算法的表达与计算机的具体高级语言形式差距较大，不够严谨，通常是用于介绍求解问题的一般算法。

2 用伪代码表示

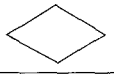
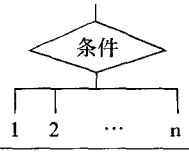
伪代码是一种介于自然语言与计算机语言之间的算法描述方法，它忽略高级程序设计语言中一些严格的语法规则与描述细节，因此，它比程序设计语言更容易描述和被人理解，而比自然语言更接近程序设计语言。虽然不能直接执行但很容易被转换成高级语言，它结构性较强，比较容易书写和理解，修改起来也相对方便。其特点是不拘泥于语言的语法结构，而着重以灵活的形式表现被描述对象。它利用自然语言的功能和若干基本控制结构来描述算

法。伪代码没有统一的标准，可以自己定义，也可以采用与程序设计语言类似的形式。

3. 用传统流程图描述算法

流程图也叫框图，是使用各种几何图形、流程线及文字说明来描述计算过程的框图。它是一种传统的算法表示法，利用几何图形的框来代表各种不同性质的操作，用流程线来指示算法的执行方向。由于流程图简单直观，使设计者的思路表达得清楚易懂，它又便于检查修改，所以应用广泛，特别是在早期程序语言设计阶段，只有通过流程图才能简明地表述算法，流程图成为程序员们交流的重要手段，直到结构化的程序设计语言出现，程序员对流程图的依赖才有所降低。如表 1－1 所示是用传统流程图描述算法时常用的符号。

表 1－1　流程图常用符号

流程图符号	含　　义
	数据输入/输出框，用于表示数据的输入和输出
	处理框，描述基本的操作功能，如“赋值”操作、数学运算等
	两分支判断框，根据框中给定的条件是否满足，选择执行两条路径中的一条
	开始/结束框，用于表示算法的开始与结束
	连接符，用于连接流程图中不同地方的流程线
	流程线，表示流程的路径和方向
	多分支判断框，根据框中的“条件值”，选择执行多条路径中的一条
	注释框，框中内容是对某部分流程图做的解释说明

用流程图描述算法时，一般要注意以下几点：

(1) 应根据解决问题的步骤从上至下顺序地画出流程图，各图框中的文字要尽量简洁。

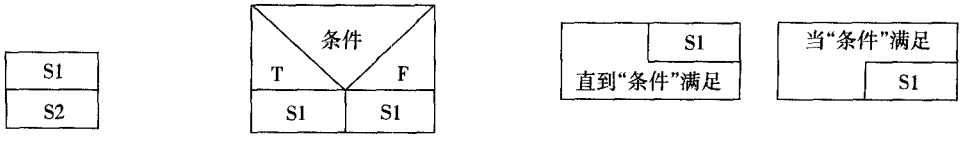
(2) 为避免流程图的图形显得过长，图中的流程线要尽量短。

(3) 用流程图描述算法时，流程图的描述可粗可细，总的原则是：根据实际问题的复杂性，流程图达到的最终效果应该是，依据此图就能用某种程序设计语言实现相应的算法（即完成编程）。

4. N－S 结构化流程图

N－S 结构化流程图（简称 N－S 图）是由美国人 I. Nassi 和 B. Shneiderman 共同提出的，其主要特点是取消了流程线，全部算法由一些基本的矩形框图顺序排列组成一个大矩形来表示，即不允许程序任意转移，而只能顺序执行，从而使程序结构化。N－S 图也是流程图的

一种很好的表示方法，三种基本控制结构的N-S图如图1-4所示。



(a) 顺序结构　　(b) 选择结构　　(c) 循环结构

图1-4 N-S图的三种基本控制结构

图1-4中“S1”或“S2”既可以是简单功能的操作（如数据赋值、数据的输入或输出等），也可以是三种基本控制结构中的一种。

在本教材中有一些简单的例题，比较详细地示范了使用什么方法来进行算法的描述，请大家认真体会一下这些常用算法的描述方法。

1.2.2 算法设计的要求

通常设计一个“好”的算法应考虑达到以下目标。

(1) 正确性（correctness）：算法应当满足具体问题的需求。通常一个大型问题的需求，要以特定的规格说明方式给出，而一个实习问题或练习题，往往就不那么严格，目前多数是用自然语言描述需求，它至少应当包括对于输入、输出和加工处理等的无歧义性的描述。设计或选择的算法应当正确地反映这种需求，否则，算法的正确与否的衡量准则就不存在了。

“正确”一词的含义在通常的用法中有很大差别，大体可划分为以下四个层次，如表1-2所示。

表1-2 算法正确性的四个层次

A. 程序不含语法错误	max (int a, int b, int c) {if (a>b) {if (a>c) return c; else return a;} }
B. 程序对于几组输入数据能够得出满足规格说明要求的结果	max (int a, int b, int c) {if (a>b) {if (a>c) return a; else return c;} } /* 8, 6, 7 *//* 9, 3, 2 */
C. 程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果	max (int a, int b, int c) {if (a>b) {if (a>c) return a; else return c;} else {if (b>c) return b; else return c;} }
D. 程序对于一切合法的输入数据都能产生满足规格说明要求的结果	—

显然，达到D层意义下的正确是极为困难的，所有不同输入数据的数量大得惊人，逐一验证的方法是不现实的。对于大型软件需要进行专业测试，而一般情况下，通常以C层意义的正确作为衡量一个程序是否合格的标准。

(2) 可读性（readability）：算法主要是为了人们的阅读与交流，其次才是机器执行。可

读性有助于人们对算法的理解，并且晦涩难懂的程序易于隐藏较多错误，难以调试和修改。

(3) 稳健性（robustness)：当输入数据非法时，算法也能适当地作出反应或进行处理而不会产生莫名其妙的输出结果。例如，一个求凸多边形面积的算法，是采用求各三角形面积之和的策略来解决问题的，当输入的坐标集合表示的是一个凹多边形时，不应继续计算，而应报告输入出错。并且，处理出错的方法应是返回一个表示错误或错误性质的值，而不是打印错误信息或异常，并中止程序的执行，以便在更高的抽象层次上进行处理。

(4) 效率与低存储量需求：效率指的是算法执行时间。对于解决同一问题的多个算法，执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。两者都与问题的规模有关。任何一个算法都要求一定量的内存，当然算法所需要的内存越少越好。实际上，算法的效率与内存消耗量都与问题的规模（即数据量）有关。因此常用时间复杂度和空间复杂度来衡量算法的效率和内存消耗量。设计出一个好的算法不是一蹴而就的事情，设计者需要具有良好的算法设计习惯，具体如表 1-3 所示。

表 1-3　两个算法效率比较

问　　题	算法一	算法二

在3个整数中求最大者	max(int a,int b,int c)
	{if (a>b)
	{if(a>c) return a;
	else return c;
	} else{if(b>c) return b;
	else return c;}
	/*无需额外存储空间，只需两次比较*/
		max(int a[3])
		{int c,int i;
		c=a[0];
		for(i=1;i<3;i++)
		if (a[i]>c) c=a[i];
		return c;}
		/*需要两个额外的存储空间，两次比较，至少一次赋值*/
		/*共需5个整型数空间*/

求100个整数中最大者	同上的算法难写，难读	max(int a[100])
		{int c,int i;
		c=a[0];
		for(i=1;i<100;i++)
		if (a[i]>c) c=a[i];
		return c;}
		/*共需102个整型数空间*/

1.2.3　算法效率

一种数据结构的优劣由实现其各种运算的算法来具体体现，对数据结构的分析实质上就是对实现运算算法的分析，除了要验证算法是否正确解决该问题之外，需要对算法的效率作性能评价。在计算机程序设计中，对算法进行分析是十分重要的。通常对于一个实际问题的解决，可以提出若干个算法，那么如何从这些可行的算法中找出最有效的算法呢？或者有了一个解决实际问题的算法，我们如何来评价它的好坏？等等，这些问题需要通过算法分析来确定。因此，算法分析是每个程序设计人员都应该掌握的技术。评价算法的标准很多，评价一个算法的好坏主要看这个算法所占用机器资源的多少，而这些资源中时间代价与空间代价是两个主要的方面，通常是以算法执行所需的机器时间和所占用的存储空间来判断一个算法的优劣。

算法执行时间需通过依据该算法编制的程序在计算机上运行时所消耗的时间来度量。而

度量一个程序的执行时间通常有两种方法。

(1) 事后统计的方法：因为很多计算机内部都有计时功能，有的甚至可精确到毫秒级，不同算法的程序可通过一组或若干组相同的统计数据来分辨优劣。但这种方法有两个缺陷：一是必须先运行依据算法编制的程序；二是所得时间的统计量依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优劣。因此人们常常采用事前分析估算的方法。

(2) 事前分析估算的方法：一个用高级程序语言编写的程序在计算机上运行时所消耗的时间取决于下列因素：

① 依据的算法选用何种策略；

② 问题的规模，例如求 100 以内还是 1000 以内的素数；

③ 书写程序的语言，对于同一个算法，实现语言的级别越高，执行效率就越低；④ 编译程序所产生的机器代码的质量；

⑤ 机器执行指令的速度。

显然，同一个算法用不同的语言实现，或者用不同的编译程序进行编译，或者在不同的计算机上运行时，效率均不相同。这表明使用绝对的时间单位衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，可以认为一个特定算法“运行工作量”的大小，只依赖于问题的规模（通常用整数量 n 表示），或者说它是问题规模的函数。

一个算法是由控制结构（顺序、分支和循环三种）和原操作（指固有数据类型的操作）构成的，则算法时间取决于两者的综合效果。为了便于比较同一问题的不同算法，通常的做法是，从算法中选取一种对于所研究的问题（或算法类型）来说是基本操作的原操作，以该基本操作重复执行的次数作为算法的时间量度。这种衡量效率的办法所得出的不是时间量，而是一种增长趋势的度量。它与软硬件环境无关，只暴露算法本身执行效率的高低。

【例 1-3】 在如下所示的两个 $n \times n$ 矩阵相乘的算法中：

```
for(i = 1;i <= n; ++i)
      for(j= 1;j <= n; ++j){
          c[i][j] = 0;
          for(k== 1;k<= n; ++k)
              c[i][j] += a[i][k]*b[k][i];}
```

“乘法”运算是“矩阵相乘问题”的基本操作。整个算法的执行时间与该基本操作（乘法）重复执行的次数 n^3 成正比，记作 $T(n) = O(n^3)$。

一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数 f(n)，算法的时间量度记作：$T(n) = O(f(n))$。

它表示随问题规模 n 的增大，算法执行时间的增长率和 f(n)的增长率相同，称作算法的渐近时间复杂度（asymptotic time complexity），简称时间复杂度。

显然，被称作问题的基本操作的原操作应是其重复执行次数和算法的执行时间成正比的原操作，多数情况下它是最深层循环内的语句中的原操作，它的执行次数和包含它的语句的频度相同。

语句的频度（frequency count）指的是该语句重复执行的次数，例如，在下列三个程序段中：

```
(a) { ++ x;s= 0;}
```

```
(b) for(i=1;i<=n;++i){++x;s+=x;}
(c) for(i=1;i<=n;++i)
        for(k=1;k<=n;++k){ ++x;s+=x;}
```

含基本操作“x增1”的语句的频度分别为1、n和n^2，则这三个程序段的时间复杂度分别为O(1)，O(n)和$O(n^2)$，分别称为常量阶、线性阶和平方阶。算法还可能呈现的时间复杂度有对数阶O(logn)，指数阶$O(2^n)$等。不同数量级时间复杂度的性状如图1-5所示。从图中可见，应该尽可能选用多项式$O(n^k)$的算法，而不希望用指数阶的算法。一般情况下，对一个问题（或一类算法）只需选择一种基本操作来讨论算法的时间复杂度即可，有时也需要同时考虑几种基本操作，甚至可以对不同的操作赋以不同权值，以反映执行不同操作所需的相对时间，这种做法便于综合比较解决同一问题的两种完全不同算法。

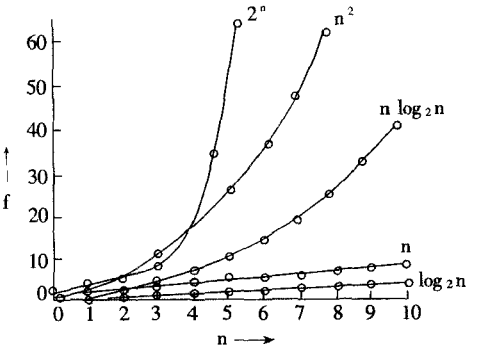


图1-5 不同数量级的时间复杂度

由于算法的时间复杂度考虑的只是对于问题规模n的增长率，则在难以精确计算基本操作执行次数（或语句频度）的情况下，只需求出它关于n的增长率或阶即可。

【例1-4】 在下列程序段中：

```
for(i=2;i<=n;++i)
    for(j=1;j<=i-1;++j){
        ++x;
        a[i][j]=x;}
```

语句++x执行次数关于n的增长率为n^2，它是语句频度表达式$(n-1)(n-2)/2$中增长最快的项。

有时，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同。一种解决的办法是计算它的平均值，即考虑它对所有可能的输入数据集的期望值，此时相应的时间复杂度为算法的平均时间复杂度。另一种更可行也更常用的办法是讨论算法在最坏情况下的时间复杂度，即分析最坏情况以估算算法执行时间的一个上界。

实践中可以把事前估算和事后统计两种办法结合起来使用。以两个矩阵相乘为例，若上机运行两个10×10的矩阵相乘，执行时间为12ms，则由算法的时间复杂度$T(n)=O(n^3)$可估算两个31×31的矩阵相乘所需时间大致为$(31/10)^3\times12\text{ms}\approx358\text{ms}$。

1.2.4 算法的空间需求

关于算法的存储空间需求，类似于算法的时间复杂度，通常采用空间复杂度作为算法所